Response Time Analysis of Systems with Real-Time and non Real-Time Processing

Dietmar PRISCHING
AVL List GmbH
Graz, Austria

Bernhard RINNER
Institute for Technical Informatics, Graz University of Technology
Graz, Austria

ABSTRACT

Most real-world automation systems (AS) require both real-time (RT) as well as non real-time (NRT) functionality. Modern AS are more frequently realized as a single platform solution in order to reduce their development time. This trend is driven by the advances in processor technology. The interoperability between RT and NRT processing is an important parameter in a single platform solution.

This paper focuses on the modeling of the RT-NRT interoperability based on response time analysis (RTA) which is extended by the context switches between RT and NRT. Our extended RTA is now able to derive the NRT load distribution in dependency on the RT load, and important interoperability parameter such as NRT utilization and maximum NRT suspension time can be computed. We apply our model on a complex AS system (PUMA Open) that uses dedicated operating systems for RT and NRT processing, respectively.

Keywords: response time analysis, interoperability, embedded systems, automation system, fixed-priority scheduling

1 INTRODUCTION

Automation systems (AS) are an important class within the real-time systems (RTS). A key requirement is that their computing systems must react within guaranteed time windows to events in the environment. Predictability and dependability are, therefore, essential for an AS. Most real-world AS, however, include also functions that do not require real-time guarantees, i.e., they are comprised of a *real-time (RT)* part and a *non real-time (NRT)* part. In a typical AS, the RT part is responsible for control, data acquisition, signal conditioning and monitoring. The NRT part is responsible for data post processing, visualization, data persistence, system parameter settings and, especially, for the graphical user interfaces (GUI).

Modern AS are more frequently realized as *a single platform solution*, because the development time is smaller than the development time of the RT and NRT parts on different platforms. This trend is also driven by the advances in processor technology. Processing power has increased considerably over the last years, and a single processor may now offer sufficient power to run AS with RT and NRT parts.

An important parameter for the combination of NRT and RT processing is *interoperability,* which is defined as the ability to run the NRT part along with the RT part and vice-versa [7], [12]. An important requirement for a RT-NRT combination is that the RT part is preferred over the NRT part, and the NRT part does not influence the RT part. There are solutions for combining RT and NRT processing commercially available. These solutions are mainly based on extending general-purpose operating systems with real-time features or by combining dedicated operating systems for NRT and RT processing. However, these solutions lack in considering the influence of the context switch between the RT and NRT part to the RT timing properties. Furthermore, the performance of the NRT part in dependency on the RT hasn't also been clarified. Particularly, the maximum suspension of the NRT computation can affect the NRT performance severely.

This paper focuses on a model to calculate performance metrics of complex systems that consist of a combination of RT and NRT processing. The performance metrics are (i) the response times of RT tasks, (ii) the CPU utilization of the RT processing and (iii) NRT timing properties. The modeling is based on a *response time analysis (RTA)* derived from Burns [3], [4] and Bernat [1]. RTA is an effective, simple and flexible technique that allows the modeling of most aspects of fixed priority real-time systems. We have extended this approach to quantify the interoperability of RT and NRT processing. Therefore, we include the overhead of RT and NRT processing as well as the overhead of the RT processing itself into the RTA formulation. Furthermore, we show that RTA can be used to retrieve the NRT computation distribution in dependency on a RT processing load.

Our approach is demonstrated on the automation system (AS) PUMA Open, which has been developed over the last few years by more than 50 engineers and that is released at the market since 2001. It is targeted for the test of engines, transmissions and power trains. This AS combines both RT and NRT computing on the same platform and has been built on the general operating system (OS) ©Microsoft Windows and its real-time extension INtime

The remainder of this paper is organized as follows. Section 2 briefly summarizes related work on response time analysis (RTA). Section 3 introduces our analytic model which introduces RT and NRT interoperability into the RTA formulation. Section 4 describes the implementation of our modeling approach and presents results. Finally, Section 5 concludes this paper with a summary and a discussion on further work.

2 RESPONSE TIME ANALYSIS (RTA)

Significant for our work are (i) multiple invocations of the tasks (ii) idle time at level i and (iii) kernel overhead.

Worst Case Response Times Analysis

[2] applies an engineering approach to calculate the *worst case response time* (R_i) for each task (τ_i,) at the *critical instant* [11] (all tasks are released together). R_i can be computed with the following equation. R_i is the smallest $\omega \geq 0$ such that:

$$\omega = C_i + I_i(\omega) \qquad \textbf{(2-1)}$$

C_i…Computation time of task τ_i

where $I_i(\omega)$ is the interference of tasks of higher priority than τ_i during interval $[0,\omega)$ and is given by:

$$I_i(\omega) = \sum_{\tau_j \in hp(\tau_i)} \left\lceil \frac{\omega}{T_j} \right\rceil \cdot C_j$$

T_j…Cycle time of task τ_j

Definition 2-1: (Priority relations) *$hp(\tau_i)$ is the set of tasks of higher priority than τ_i; $hep(\tau_i)$ is the set of tasks of higher or equal priority than task τ_i, and $hep(\tau_{ij})$ is set of tasks of higher or equal priority than task τ_i also including task τ_i.*

Finish Time (F_i) of the kth Task Invocation

There may be some time the processor is used by lower priority tasks between the finalization of the first invocation and the start of the second one. This time is called the *idle time* at level i (δ_i). Formally:

Definition 2-2: *($\delta_i(\omega)$) The idle time at level i at ω is the amount of time the processor can be used by tasks of lower priority than τ_i during period of time $[0,\omega)$.*

The amount of idle time at the start of each task invocation (*start time* (S_i)) is of special interest. Thus, it can also be written as $\delta_i(k) = \delta_i(S_i(k))$. The computation of *finish time* $F_i(2)$ is based on the fact that this time is equivalent to the worst case response time of a task with the same priority as task τ_2 and a computation time $C = 2C_2 + \delta_2(2)$. Thus, the worst case finish time of the second invocation of task $\tau_i \in \prod$ ($\Pi = \{\tau_i = (T_i, D_i, C_i) \mid 1 \leq i \leq N\}$), is the smallest $\omega \geq 0$ such that:

$$\omega = 2C_i + \delta_i(S_i(2)) + I_i(\omega) \qquad \textbf{(2-2)}$$

This formulation can be extended for the kth invocation.

$$\omega = kC_i + \delta_i(k) + I_i(\omega) \qquad \textbf{(2-3)}$$

Computing the Idle Time $\delta_i(t)$

The computation of $\delta i(t)$ is more complex because it can not be computed directly. We will show this with the following counter example (see Table 2-1). Someone can assume that:

$$\delta_i(t) = t - \sum_{\tau_j \in hep(\tau_i)} \left\lceil \frac{\omega}{T_j} \right\rceil \cdot C_j .$$

If we carry out this equation we get $\delta_{T3}(10) = 3.5$. However, $\delta_{T3}(10) = 4.5$ due to the fact that the second invocation from T2 can't be accounted full.

Table 2-1: Example task set.

Task	T	C	Priority
T1	5	1	1
T2	9	2	2
T3	10	0.5	3

So, the computation of $\delta_i(t)$ is based on the observation that the amount of idle time between $[0,t)$ is equal to the maximum computation time a single task, running at a lower priority than τ_i, could use during $[0,t)$. To compute this value it is assumed that the task set is made up only of the tasks of higher or equal priority than task τ_i plus a virtual task, τ_v, with lower priority than τ_i. Task τ_v will consume all unused computation time of tasks of higher or equal priority than task τ_i. To compute the amount of idle time at level i between $[0,t)$, task τ_v has a period and a *deadline* (D_v) equal to the time t: $\tau_v = (T_v = t, D_v = t, C_v)$. The maximum time the processor can be used by tasks of lower priority than τ_i is the maximum computation time, C_v, that makes task τ_v meet its deadline. Formally:

$$\boldsymbol{\delta_i(t) = \max\{C_v \mid \tau_v \text{ is schedulable}\}}$$

Schedulable means that a certain C_v is available within the period t, and the scheduling test is performed by solving the next equation for ω and checking whether $\omega \leq D_v$:

$$\omega = C_v + I_i^{\bullet}(\omega) \qquad \textbf{(2-4)}$$

where the amount of interference at level i including task τ_i, denoted by $I_i^{\bullet}(\omega)$, is given by:

$$I_i^{\bullet}(\omega) = \sum_{\tau_j \in hep(\tau_{ii})} \left\lceil \frac{\omega}{T_j} \right\rceil \cdot C_j \qquad \textbf{(2-5)}$$

As described above, this equation can be computed by solving a recurrence relation.

Kernel Overhead

Simple scheduling models ignore the *operating system* (OS) software kernel behavior. Context switch times and queue manipulations are, however, significant and cannot be neglected. When a software kernel is used, models of the actual behavior are needed. Without these models, excessively pessimistic overheads must be assumed. *Context switch* times can be accounted for by adding their cost to the task that causes the context switch. For periodic tasks, the cost of the insertion into the delay queue and switching back to the lower-priority task that has been preempted is, however, not necessarily constant. More about task scheduling, delay queue, run queue can be found in, e.g., [10], [14]. The interrupt handler for the clock does the manipulation of the delay queue. The manipulation cost may depend on the size of the delay queue, i.e., on the number of periodic tasks in the application. To model adequately the delay queue manipulations that occur in the clock interrupt handler, it is necessary to address directly the overhead caused by each periodic task. It may be possible to model the clock interrupt handler using two parameters: C_{CLK} (the overheads occurring on each interrupt assuming that tasks are on the delay queue but that none are removed), and C_{PER} (the cost of moving one task from the delay queue to the run queue). Each periodic task now has a fictitious task (fpt is set of fictitious periodic tasks) with the same period T but with computation time C_{PER} [2].

$$\omega = C_i + \sum_{j \in hp(i)} \left\lceil \frac{\omega}{T_j} \right\rceil \cdot C_j + \left\lceil \frac{\omega}{T_{CLK}} \right\rceil \cdot C_{CLK} + \sum_{f \in fpt} \left\lceil \frac{\omega}{T_f} \right\rceil \cdot C_{PER} \qquad \textbf{(2-6)}$$

fpt…fictitious task set

3 MODELING RT-NRT INTEROPERABILITY

Performance metrics of interest in this work are (i) the response times of RT tasks, (ii) the CPU utilization of a RT load and (iii) NRT timing properties. We start from a process model made up of n RT tasks and a single NRT task. When no RT task is ready to execute, the lowest priority task (NRT task) is scheduled. Our fixed priority model is based on the assumption of a closed system. That means that the pattern of the response times of all tasks is repeated periodically within the hyper-period **H**.

Response times

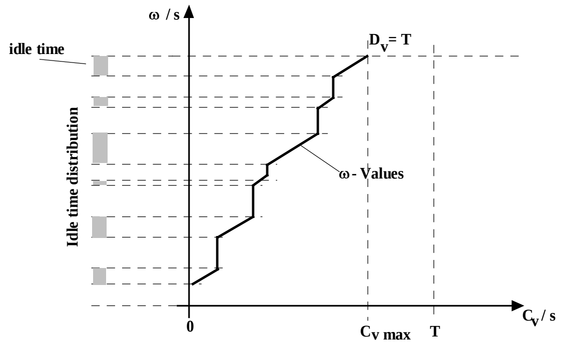

Figure 3-1: The distribution of the idle time at each level can be derived from the computed values of ω.

First we describe the exact computation of the timing properties of a specific RT-processing that doesn't consider interoperability at all. We can calculate the exact *idle time distribution* within the cycle (T) of the hyper-period **H** for each level i. The computing of the idle time distribution for a individual level i ($\delta_i(0,T)$) is based on Eq. (2-4) [1]. C_v is incremented from 0 to T until the condition $\omega \leq D_v$ fails (see Figure 3-1). The computed values of ω can be used to derive the idle time distribution. Each monotonic rise of ω with regard to C_v corresponds to an idle time, and each discontinuous step of ω corresponds to an interference at level i (see Figure 3-1). This computation is carried out for each level i. Now, multiple invocations for each task τ_i can be calculated with the idle time distributions $\delta_i(0,T)$. The worst case finish time of multiple invocations ($F_i(k)$) is based on Eq. (2-3). In order to introduce context switch times in our approach, we also need the start times of the multiple task invocations. We can compute these start times with a simple modification of Eq. (2-3). We simply set the kth computation time of task τ_i as the smallest possible time unit that is used in the model. Therefore, the computation of start time $S_i(k)$ is based on the fact that this time is equivalent to the worst case response time of a task with the same priority as task τ_i and a computation time C = 1(smallest time unit) + (k-1)C_i + $\delta_i(k)$.

$$\omega = 1 + (k-1)C_i + \delta_i(k) + I_i(\omega) \qquad \textbf{(3-1)}$$

To summarize, we are now able to compute (i) the exact idle time distribution at each level i $\delta_i(0,T)$ within the hyper-period, and (ii) the start and finish time for the multiple invocations within the hyper-period.

Next, we describe a method for computing the timing properties of a RT-processing that also considers interoperability with NRT. We start with the approach from Eq. (2-6) [2]. The last term in this equation represents the overhead caused by task context switches, and C_{PER} is the context switch time. In general, C_{PER} depends on whether a context switch between NRT and RT, RT and RT across different processes and RT and RT within the same process takes place (thread vs. process see [14]). For further clarification we denote a OS context switch as *NRT – RT* and a thread context switch within the same process as *RT-RT(SP)* and a thread context switch across different processes as *RT-RT(DP).* The time for a NRT-RT is defined as $C_{NRT\text{-}RT}$, for RT-RT(SP) as $C_{RT\text{-}RT(SP)}$ and for RT-RT(DP) as $C_{RT\text{-}RT(DP)}$ (Examinations about modeling context switch time see [5]). With the calculated start and finish times of the tasks it is possible the retrieve the kind of context switch that occurs for a certain invocation of a task τ_i. This happens as follows. In Figure 3-2 we see the invocations of a task set that consist of two RT tasks (*Task1*, *Task2*). At an invocation of *Task1* an OS switch happens (NRT – RT) happens. Furthermore also a thread context switch between the NRT task and *Task1* also occurs. We define this as a RT-RT(DP). At the finish of *Task1* a thread context switch to *Task2* happens. In dependency whether *Task1* and *Task2* belong to the same or different processes a RT-RT(SP) or RT-RT(DP) occurs. The information whether or not a context switch occurs prior to a certain task invocation can be represented by step functions **Φ** ($\Phi^{NRT\text{-}RT}$, $\Phi^{RT\text{-}RT(SP)}$, $\Phi^{RT\text{-}RT(DP)}$). In Table 3-1 we see these step functions for the above example under the assumption *Task1* and *Task2* belongs to the same process.

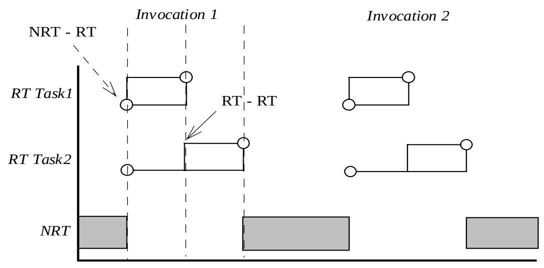

Figure 3-2: Schematic thread invocations

Table 3-1: Step function fort he example task set.

Invocation (k)	*Task1*		
	$\Phi_{Task1}^{NRT-RT}(k)$	$\Phi_{Task1}^{RT-RT(SP)}(k)$	$\Phi_{Task1}^{RT-RT(DP)}(k)$
1	1	0	1
2	1	0	1
	Task2		
	$\Phi_{Task2}^{NRT-RT}(k)$	$\Phi_{Task2}^{RT-RT(SP)}(k)$	$\Phi_{Task2}^{RT-RT(DP)}(k)$
1	0	1	0
2	0	1	0

The interference caused by the NRT-RT context switch overhead at level i including context switches of task i, denoted by $I_i^{NRT-RT}(\omega)$, is given by (for hep(τ_{ij}) see Definition 2.1):

$$I_i^{NRT-RT}(\omega) = \sum_{\tau_j \in hep(\tau_{ij})} \sum_{k=0}^{\left\lceil \frac{\omega}{T_j} \right\rceil} \Phi_j^{NRT-RT}(k) \cdot C_{NRT-RT}$$

In the same way, the interference caused by the RT-RT(SP) and RT-RT(DP) respectively overhead at level i including context switches of task i, denoted by , is given by:

$$I_i^{RT-RT(SP)}(\omega) = \sum_{\tau_j \in hep(\tau_{ij})} \sum_{k=0}^{\left\lceil \frac{\omega}{T_j} \right\rceil} \Phi_j^{RT-RT(SP)}(k) \cdot C_{RT-RT(SP)}$$

and

$$I_i^{RT-RT(DP)}(\omega) = \sum_{\tau_j \in hep(\tau_{ij})} \sum_{k=0}^{\left\lceil \frac{\omega}{T_j} \right\rceil} \Phi_j^{RT-RT(DP)}(k) \cdot C_{RT-RT(DP)}$$

The interferences from context switch between NRT and RT, RT-RT(SP) and RT-RT(DP) can be summarized and we write:

$$I_i^{CS} = I_i^{NRT-RT} + I_i^{RT-RT(SP)} + I_i^{RT-RT(DP)}$$

Therefore, NRT-RT interference is included into the response time calculation at the critical instant, with the following extension of Eq. (2-1):

$$\omega = C_i + I_i(\omega) + I_i^{CS}(\omega) \qquad \textbf{(3-2)}$$

Similarly, the NRT-RT interference is included into the idle time computation by extending Eq. (2-4):

$$\omega = C_V + I_i^{\bullet}(\omega) + I_i^{CS}(\omega) \qquad \textbf{(3-3)}$$

Modeling the RT – NRT interoperability extends the basic formulations of the response time analysis (RTA). Therefore, offsets, sporadic tasks, blocking factors and release jitter formulations can also be included in our RTA formulation. These extensions are, however, not discussed in this paper.

CPU Utilization

Given a set Π of n periodic tasks, the processor utilization factor U is the fraction of processor time spent in the execution of the task set. Since Ci/Ti is the fraction of processor time spent in executing task τ_i, the utilization factor for n tasks is given by [6]:

$$U = \sum \frac{C_i}{T_i} \qquad \textbf{(3-4)}$$

With the previous results the rate of certain context switches is knows. So we can denote f_{NRT-RT} as the frequency of NRT-RT, with $f_{RT-RT(SP)}$ as the frequency of RT-RT(SP) and with $f_{RT-RT(DP)}$ as the frequency of RT-RT(SP). So, with an extension of the formulation from the CPU utilization for a specific RT-processing is given by:

$$U_\Pi = \sum \frac{C_j}{T_j} + f_{NRT-RT} \cdot C_{NRT-RT} + f_{RT-RT(SP)} \cdot C_{RT-RT(SP)} + f_{RT-RT(DP)} \cdot C_{RT-RT(DP)} \qquad \textbf{(3-5)}$$

NRT Properties

The following properties of the RT-NRT interoperability can be derived from the above calculations.

- The $\delta_i(0,T)$-distribution at the lowest priority level i is equal to the NRT computation distribution.
- The response time of the lowest priority task at the critical instant corresponds with the longest suspension of NRT computation.
- f_{NRT-RT} is a crucial factor for NRT-RT interoperability overhead. NRT update-functionality (E.g. graphic) is disturbed through a higher f_{NRT-RT}. Thus the RT-NRT interoperability performance isn't 100% at high f_{NRT-RT} (see [12],[13])

4 IMPLEMENTATION AND CASE STUDY

Computing the timing properties

The main steps for computing the timing properties for a RT-NRT processing can be summarized as follows:

1. First the timing metrics of a RT-processing that does not consider interoperability are calculated.
2. Second, with the calculated start and finish times of the tasks the step functions ((Φ^{NRT-RT}, $\Phi^{RT-RT(SP)}$, $\Phi^{RT-RT(DP)}$) are determined.
3. In a third step, with our extended equations (see Eq. (3-2) and Eq. (3-3)) start and finish times of the tasks are calculated that consider NRT-RT, RT-RT(SP) and RT-RT(DP). Howsoever, these results can change the step functions and so step 2 and step 3 have to be repeated until the step function does not change between the iterations.
4. Finally, the CPU utilization from the RT load based on Eq. (3-5) is calculated as well as the NRT properties can be determined based on the timing metrics results (see chapter 3, NRT Properties).

PUMA Open Automation System

We have applied our process model in the automation system (AS) PUMA Open. PUMA Open is targeted for the design and test of engines, transmissions and power trains. It is a large and complex AS running under the operating systems Windows NT/2000/XP for NRT processing and INtime for RT processing, respectively. Its basic functionality includes safety monitoring, data acquisition and storage as well as control and automation functions. Data acquisition is performed at frequencies up to 10 kHz. Furthermore, PUMA Open is required to operate complex control loops within 1 ms response time. Due to these requirements, it is important to understand and model all mechanism of its RT processing as well as predict the timing behavior in the resolution of microseconds.

In [13] we have evaluated the Windows-INtime interoperability. For a PENTIUM II (434 MHz) configuration the OS context switch time (C_{NRT-RT}) was determined as 3 µs. Furthermore a RT-RT(DP) is also determined with 3 µs and a RT-RT(SP) with 2 µs. Thus, a high context switch rate has a significant influence on the system performance.

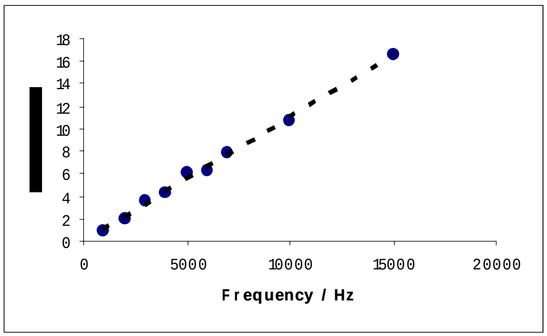

Figure 4-1: The relative CPU usage caused by Windows–INtime context switches at different rates.

Further Windows-INtime results are (i) that a suspension of Windows for more than 7 ms affects the GUI is severely, (ii) a suspension of 10ms disturbs the network [15] and (iii) that the Windows-INtime interoperability performance overhead may exceed 10 % at high OS context switch rates. To monitor a representative task set of a RT-processing we have implemented the *ProfileAnalyzer* [13]. The ProfileAnalyzer is a tool that delivers accurate measurements of thread context switches and thread execution times.

Example Task Set

Recent timing requirements of the PUMA Open are data acquisition, monitoring and control with 10 kHz. So in this example, we demonstrate the applicability of our model. We work with three tasks (thread T1, T2 and T3) shown in Table 4-1 and each thread belong to an own process (thread vs. process see [14]). Figure 3-1 depicts the result of multiple invocations that doesn't consider interoperability and
Figure 4-3 and Table 4-3 shows the result of our model that includes the interoperability overhead. Without the context switches T1, T2 and T3 keep its deadlines (see **Error! Not a valid bookmark self-reference.**). However, when including the context switch times T3 misses its deadline (see Table 4-3).
Table 4-2 shows the calculated timing properties.

Table 4-1: Example Task Set 2

Process	Task	T / µs	C / µs	D / µs	Priority
A	T1	100	20	100	1
B	T2	200	50	100	2
C	T3	400	20	100	3

Figure 4-3 and Table 4-3 shows the result of our model that includes the interoperability overhead. Without the context switches T1, T2 and T3 keep its deadlines (see **Error! Not a valid bookmark self-reference.**). However, when including the context switch times T3 misses its deadline (see Table 4-3).

Table 4-2: Calculated timing properties that don't consider interoperability.

Task	k	A / µs	S / µs	F / µs	R / µs
T1	1	0	0	20	20
	2	100	100	120	20
	3	200	200	220	20
T2	1	0	20	70	70
	2	200	220	270	70
	3	400	420	470	70
T3	1	0	70	90	90
	2	400	470	490	90

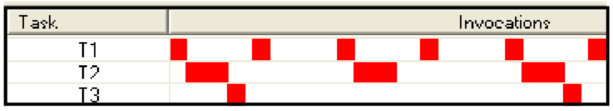

Figure 4-2: Calculated multiple invocations that don't consider interoperability.

Table 4-3: Response times including context switches.

Task	k	A / µs	S / µs	F / µs	R / µs
T1	1	0	6	26	26
	2	100	103	123	26
	3	200	206	226	26
T2	1	0	29	79	79
	2	200	229	279	79
	3	400	429	479	79
T3	1	0	82	128	**128**
	2	400	482	528	**128**

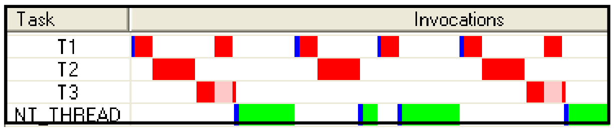

Figure 4-3: Calculated multiple invocations that consider interoperability.

The calculated CPU utilization without interoperability is U_{rt}= 50 %. At including interoperability f_{NRT-RT} follows with 7450 Hz and $f_{RT-RT(DP)}$ with 19950 Hz. Therefore, the CPU utilization with Eq. (3-5) is U_{rt}= 60.48 %. The longest suspension from NRT (NT_THREAD) is 128 µs that is far away from 7ms, that can disturb NRT. However, an f_{NRT-RT} with 7450 Hz disturbs the NRT update-functionality, so that e.g. the NRT graphic can be remarkable affected.

Representative PUMA Open task sets have more than 50 tasks. Including the interoperability results in a better fit to the measurements, especially for the CPU utilization. The deviation between measured and computed values is smaller than 1 %. We demonstrate this with the following small PUMA open counter example. In this example all RT tasks are periodic, scheduled according to the rate monotonic (RM) policy and belong to the same process. Table 4-4 shows the input timing properties (T_i and C_i) as well as with our approach calculated response times of theses tasks. Figure 4-4 depicts the computed timing behavior for this task set in a Gantt chart. Due to the complexity of this task set, the iterative calculation (see above) has to be repeated tree times, until the step functions do not change.

The context switch rate and the utilization are derived as f_{NRT-RT} = 2.9 kHz and $f_{RT-RT(SP)}$ (all tasks belong to the same process) = 10.5 kHz and so U_{Π} ~ 60%. The deviation between measured and computed values is smaller than 1%. *NT_THREAD* represents the Windows NRT task. Its maximum suspension time is computed as 3189 µs (response time of the lowest priority task) and the total computation time of the NRT task within 1s is ~ 420 ms. Given these numbers (f_{NRT-RT} is smaller than frOm example 1) we can conclude that the Windows performance is only slightly affected.

Table 4-4: Example task set 3 (PUMA Open).

τ_i	T_i [ms]	C_i [µs]	R_i [µs]
1	0.2	30	30
2	0.5	40	70
3	1	90	170
4	2	260	500
5	2	95	671
6	4	100	773
7	8	160	969
8	10	17	988
9	16	50	1242
10	20	116	1360
11	32	11	1373
12	50	80	1489
13	100	280	1883
14	200	34	1919
15	320	134	2728
16	500	55	2785
17	1000	200	3189

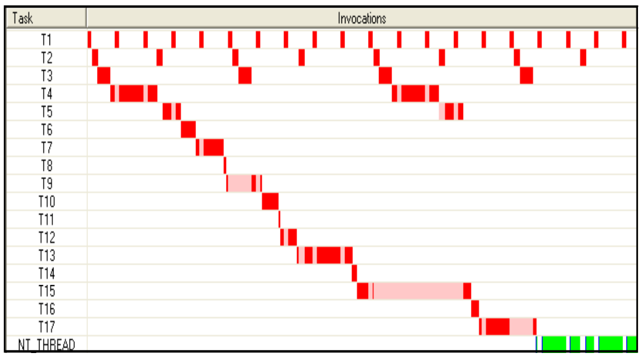

Figure 4-4: Computed timing behavior for the example task set 3 represented as Gantt charts.

5 DISCUSSION

We have presented a process model for computing the timing properties of complex systems with real-time (RT) and non real-time (NRT) processing. Our modeling technique is based on the response time analysis (RTA). We have extended this approach to quantify the interoperability of RT and NRT processing. Therefore, we include the overhead of RT and NRT processing as well as the overhead of the RT processing itself into the RTA formulation for the computation of response times of task. Furthermore, we show that the RTA formulation can be used to obtain the CPU utilization of a RT load and NRT timing properties that are important for the development and operation of most AS.

Our approach was demonstrated on the automation system (AS) PUMA that combines both RT and NRT processing on the same target platform and it is based on the general purpose OS Microsoft® Windows NT/2000/XP and its real-time extension INtime. The small deviation between the computed and the measured timing properties in our PUMA Open test cases demonstrates the feasibility of our approach.

The computation of each value of the idle time distribution $\delta_i(0,T)$ has a similar complexity to the computation of R_i. As ongoing and future work we optimize and speed-up the idle time computation. Furthermore, we want to take offsets and sporadic tasks into consideration for our model. Finally, we will include a formulation for interrupts that are executed in the context of the pre-empted task.

REFERENCES

[1] BERNAT. G., 2002 Response Time Analysis of Asynchronous Real-Time Systems, University of York

[2] BURNS, A., 1994: Preemptive Priority-Based Scheduling: An Appropriate Engineering Approach, Advances in Real Time Systems, 1994 225-248.

[3] AUDSLEY, N., C.; BURNS, A., 1993 Applying New Scheduling Theory to Static Priority Pre-emptive Scheduling. Software Engineering Journal

[4] BURNS, A., WELLINGS. A., 2001: Real-Time systems and programming languages. Addision Wesley 3rd edition, 2001

[5] BUSQUETS, M., WELLINGS A., 1996: Adding Instruction Cache Effect to Schedulability Analysis of Preemptive Real Time Systems. University of York

[6] BUTTAZZO, G. C., 1997: HARD REAL_TIME COMPUTING SYSTEMS, Predictable Scheduling Algorithms and Applications, London, Kluwer Academic Publishers

[7] http://www.dedicated-systems.com

[8] Real Time Magazine Real-Time Consult "INTIME 1.20"; http://www.dedicated-systems.com

[9] INtime Software Overview Guide, RadiSys

[10] KLEIMAN, S., SHAH, D., SMAALDERS, B., 1996 Programming with Threads, California

[11] LIU, C. L., LAYLAND, J. W., 1973: Scheduling algorithms for multiprogramming in a hard-real-time environment, Journal of the Association for Computing Machinery, 20(1), pp. 46-61

[12] OBENLAND, K. M., ROSEN, L. H., 2000: The Performance Trade-offs of Implementing a Large Scale Real-time Application Using the Windows NT Operating System, Proceedings of the Sixth IEEE Real Time Technology and Applications Symposium, 2000

[13] PRISCHING, D., RINNER, B., 2003: Thread-based analysis of embedded applications with real-time and non real-time processing on a single-processor platform, embedded world 2003 Congress, Nürnberg

[14] SILBERSCHATZ. A., GALVIN, P., GAGNE, G., 2000: Applied Operating System Concepts, Hohn New York, Wiley & Sons, Inc.

[15] www.tenasys.com